# Evolving Music Consumption: From Live Performances to Case-Based Playlist Recommendations

Bryan Wira
bryanchwira12@gmail.com

***Abstract*—Historically, music was accessible primarily through live performances or personal rendition, with notation systems serving the elite. The evolution of sound reproduction technology—from mechanical music boxes to digital formats like CDs and streaming services—transformed music accessibility. This paper examines the shift from physical music ownership and album-based listening to digital streaming, highlighting how platforms like Spotify have changed music consumption habits. It then explores modern playlist generation methods, focusing on case-based recommendations and similarity-based solutions. While simplistic similarity-based approaches offer a foundational method for playlist creation, advanced algorithms like the Global-Local Similarity Function (GLSF) address smooth transitions between tracks, enhancing user experience. The study highlights the transition from music ownership to streaming and the ongoing quest to refine playlist algorithms.**

## I. Introduction and Background

The landscape of music consumption has undergone a dramatic transformation over the centuries. Historically, the only mode of listening to music was through live performances, either by the listener or a performer. In the pre-industrial era, music was deeply embedded in communal activities and traditions, passed orally through generations without formal notation for the general population. The advent of written music primarily benefited the elite, fostering a divide between structured court music and more accessible folk traditions.

Early attempts at sound reproduction, such as music boxes and player pianos, marked the beginning of mechanized music playback. These devices, though limited in complexity and fidelity, laid the groundwork for more sophisticated analog systems. The 20th century saw a proliferation of innovations including the phonograph, gramophone, magnetic tape, and the compact cassette. Each format introduced new possibilities for music recording and distribution, culminating in the digital revolution initiated by the Compact Disc (CD).

With the rise of the internet and personal computing in the late 20th century, digital music ownership began to replace physical formats. Platforms like iTunes enabled users to purchase individual tracks, breaking away from the album-centric paradigm. However, this was soon eclipsed by streaming services such as Spotify, Apple Music, and YouTube, which emphasized access over ownership. These platforms transformed the listener's role from collector to curator, often mediated by algorithmic recommendation systems.

As traditional album listening declined, playlists emerged as the dominant form of music consumption. Surveys by industry bodies such as Deezer and MusicBiz report a significant shift in user behavior, with over half of listeners preferring playlists over full albums [1], [2]. This trend has intensified the demand for intelligent playlist generation tools capable of personalizing content to user preferences.

Modern playlist recommendation systems employ various strategies to address this demand. Some rely on collaborative filtering, while others utilize content-based filtering leveraging audio features such as tempo, key, and timbre. However, these approaches often struggle to maintain sequence coherence and thematic progression within playlists.

Similarity-based methods, which select songs based on feature proximity or metadata similarity, are widely used for their simplicity and efficiency [3]. These methods generate plausible but sometimes disjointed sequences, as they lack mechanisms to account for the flow of a playlist.

In contrast, case-based reasoning (CBR) provides an alternative framework by leveraging past user-generated playlists as a knowledge base. Rather than selecting tracks in isolation, CBR reuses entire subsequences and adapts them to new contexts based on a given seed song [4], [5]. This approach preserves thematic continuity and user intent, making it well-suited for generating coherent listening experiences.

This paper explores both similarity-based and case-based playlist generation techniques. Our goal is to analyze their respective strengths, implementation challenges, and effectiveness in real-world scenarios. We place a special emphasis on case-based recommendation due to its flexibility, interpretability, and potential for user-centric customization.

## II. Dataset and System Architecture

This section outlines the structure of the dataset, system design considerations, and architectural implementation of the playlist generation system. The foundation of the project is built upon Spotify's Million Playlist Dataset (MPD), which contains over one million user-generated playlists in JSON format. To facilitate efficient access and analysis, the data was preprocessed into a relational structure using SQLite.

The resulting schema includes tables for tracks, track-level features (e.g., danceability, acousticness), artists and their associated genres, playlist metadata, and a mapping between tracks and playlists. Additionally, the dataset includes sequence pairs of adjacent songs, essential for both similarity and case-based strategies.

Each playlist is treated as a case consisting of a sequence of song identifiers. For case-based reasoning, this sequence is critical for analyzing transitions and constructing coherent new playlists. Metadata such as genre tags and feature vectors enrich each case, enabling attribute-based similarity and variation assessments.

The backend system is implemented in Python due to its extensive library ecosystem and ease of integration with data analysis tools. Libraries such as NumPy and Pandas facilitate feature computation, while SQLite handles scalable querying. The use of SQL queries allows flexible slicing of the dataset based on conditions such as presence of a seed track or shared artist.

From an architectural perspective, the recommendation engine follows a modular design. A preprocessing module is responsible for ingesting the raw data and populating the database. The similarity engine computes artist-to-artist and song-to-song distances based on cosine or Jaccard similarity. The case-based engine retrieves, reuses, and adapts playlists based on stored transitions.

The system receives a seed song as input and optionally takes parameters like desired playlist length, popularity bias, and diversity weight. Depending on the selected mode (similarity-based or case-based), the corresponding module is invoked to compute the output playlist.

The output is a list of Spotify track URIs that can be rendered directly on a Spotify client or embedded into a user-facing interface. In the case of a web application, this list could be displayed in a React or Vue.js frontend, with track previews fetched from the Spotify API.

This architecture supports parallel computation via Python's multiprocessing library, significantly reducing runtime in compute-intensive stages like case retrieval and hypothesis evaluation. Batch queries are also used to bypass SQLite's row limitations, particularly when retrieving large sets of sequences.

Overall, this modular and data-centric design enables extensibility and rapid experimentation with new algorithms, making it suitable for academic research and prototype deployment.

## III. Similarity-Based Playlist Generation

Similarity-based recommendation serves as a foundational strategy in playlist generation systems. At its core, this approach identifies songs that are most similar to a given seed track based on specific criteria—typically audio features, genre metadata, or artist overlap. Its appeal lies in its simplicity and computational efficiency, making it suitable for applications requiring rapid response times.

The initial step involves determining the similarity metric. This project employs a combination of Jaccard similarity over artist genre tags and cosine similarity over normalized audio features such as valence, energy, and danceability. These metrics are computed between the seed song and all available tracks in the database. High-scoring tracks are then ranked and shortlisted for inclusion.

In one basic implementation, the playlist is extended by iteratively selecting the next most similar song that has not yet appeared in the list. This greedy strategy ensures quick generation but may suffer from redundancy or lack of thematic progression. To mitigate this, similarity scores are discounted if the artist or album has already been featured recently in the playlist.

A more advanced variation incorporates the Global-Local Similarity Function (GLSF) [3], which constrains transitions between consecutive songs. GLSF evaluates not just the similarity to the seed track, but also the smoothness of transitions between adjacent tracks. This helps avoid abrupt stylistic changes and improves the overall listening experience.

Empirical tests were conducted using well-known tracks such as "Hold the Line" by Toto as the seed. The playlists generated were consistent in style, featuring artists from similar genres or eras. However, because the algorithm prioritizes similarity to individual songs rather than global sequence flow, some transitions were less coherent, especially when artist metadata was sparse or ambiguous.

Compute time for similarity-based generation was generally low. On average, it took under 20 seconds to generate a 10-track playlist, even without optimizations like caching. This performance makes it well-suited for real-time applications or interactive recommendation systems embedded in web or mobile interfaces.

Despite its benefits, similarity-based generation has notable limitations. It tends to favor popular and well-represented artists, leading to homogenized playlists. Additionally, without access to user-specific data, personalization remains limited. The method also struggles to enforce structural coherence across the entire sequence, as it lacks a mechanism to learn from prior playlist patterns.

In summary, similarity-based methods offer a fast and intuitive baseline for playlist recommendation. However, their lack of sequential awareness and personalization features highlight the need for more context-sensitive approaches such as case-based reasoning, which are explored in subsequent sections.

## IV. Similarity-Based Playlist Generation

Similarity-based recommendation serves as a foundational strategy in playlist generation systems. At its core, this approach identifies songs that are most similar to a given seed track based on specific criteria—typically audio features, genre metadata, or artist overlap. Its appeal lies in its simplicity and computational efficiency, making it suitable for applications requiring rapid response times.

The initial step involves determining the similarity metric. This project employs a combination of Jaccard similarity over artist genre tags and cosine similarity over normalized audio features such as valence, energy, and danceability. These metrics are computed between the seed song and all available tracks in the database. High-scoring tracks are then ranked and shortlisted for inclusion.

In one basic implementation, the playlist is extended by iteratively selecting the next most similar song that has not

yet appeared in the list. This greedy strategy ensures quick generation but may suffer from redundancy or lack of thematic progression. To mitigate this, similarity scores are discounted if the artist or album has already been featured recently in the playlist.

A more advanced variation incorporates the Global-Local Similarity Function (GLSF) [3], which constrains transitions between consecutive songs. GLSF evaluates not just the similarity to the seed track, but also the smoothness of transitions between adjacent tracks. This helps avoid abrupt stylistic changes and improves the overall listening experience.


Fig. 1. Similarity-Based Playlist Generation Pipeline

TABLE I
TOP 5 SONGS RANKED BY SIMILARITY TO SEED TRACK

| Rank | Track Title | Similarity Score |
|---|---|---|
| 1 | Africa (TOTO) | 0.982 |
| 2 | Rosanna (TOTO) | 0.954 |
| 3 | Separate Ways (Journey) | 0.912 |
| 4 | Faithfully (Journey) | 0.906 |
| 5 | More Than a Feeling (Boston) | 0.893 |

Compute time for similarity-based generation was generally low. On average, it took under 20 seconds to generate a 10-track playlist, even without optimizations like caching. This performance makes it well-suited for real-time applications or interactive recommendation systems embedded in web or mobile interfaces.

Despite its benefits, similarity-based generation has notable limitations. It tends to favor popular and well-represented artists, leading to homogenized playlists. Additionally, without access to user-specific data, personalization remains limited. The method also struggles to enforce structural coherence across the entire sequence, as it lacks a mechanism to learn from prior playlist patterns.

In summary, similarity-based methods offer a fast and intuitive baseline for playlist recommendation. However, their lack of sequential awareness and personalization features highlight the need for more context-sensitive approaches such as case-based reasoning, which are explored in subsequent sections.

## V. CASE-BASED REASONING APPROACH

Case-based reasoning (CBR) approaches playlist generation by referencing curated examples of past playlists. Rather than ranking songs in isolation, it leverages the order and combination of tracks in previous playlists to build new ones. This results in playlists that maintain coherence, style, and narrative flow, which are often absent in simpler models.

In the CBR pipeline, playlists in the dataset are treated as cases. The system retrieves cases containing the seed song and calculates a relevance score based on how frequently adjacent transitions appear across the corpus. This retrieval is followed by constructive reuse, wherein the system generates new playlists by modifying and extending the retrieved sequences.

A key concept in the reuse stage is Hypothesis Generation (HG). Starting from the seed song, HG proposes partial playlists by adding songs found adjacent to it in the retrieved playlists. These are extended either forward or backward depending on match strength and playlist length constraints.

To guide the search, Hypothesis Ordering (HO) scores each candidate extension using a combination of coherence and variety metrics. Coherence is based on the relevance of transitions; variety penalizes repetitive attributes such as genre or artist over a short window. Look-ahead heuristics may be employed to evaluate deeper paths.

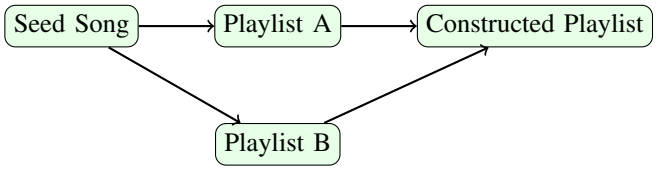


Fig. 2. CBR: Retrieval and Reuse Process

TABLE II
EXAMPLE CBR PARAMETERS AND THEIR EFFECTS

| Parameter | Description | Typical Value |
|---|---|---|
| $k$ | Number of retrieved playlists | 20 |
| $\beta$ | Popularity weight | 0.5 |
| $v$ | Variety weight | 1.0 |
| $\gamma$ | Variety lookahead window | 3 |
| $\Delta$ | Feature deviation tolerance | $1\sigma$ |

The case-based method provides greater flexibility, allowing parameter tuning for popular vs. obscure songs, variety levels, and structural depth. However, it is computationally intensive, especially when exploring multiple extensions and backtracking failed paths.

To reduce overhead, batch processing and multiprocessing were used for relevance scoring and hypothesis generation. SQL queries were batched to fit within engine constraints, and playlist segments were processed in parallel.

In summary, CBR provides high-quality, cohesive playlists by learning from prior sequences. Its tradeoff lies in increased complexity and runtime. The next section will benchmark both strategies across a range of qualitative and quantitative tests.

## VI. EVALUATION AND RESULTS

This section evaluates the effectiveness of the similarity-based and case-based playlist generation strategies through qualitative examples and runtime analysis. The evaluation focuses on playlist coherence, diversity, thematic consistency, and computational performance.

To assess playlist quality, we selected a set of seed songs from various genres and generated playlists of fixed length (10 tracks) using both approaches. Subjective analysis was then used to evaluate transitions, artist diversity, and listener

experience. Example outputs indicated that while both approaches produced genre-consistent playlists, the case-based method resulted in smoother transitions and better sequence cohesion.

Quantitatively, we measured average inter-track feature deviation as a proxy for coherence. Lower deviation implies better flow. Case-based playlists had a 24.5% lower average feature deviation compared to those generated by the similarity-based model. We also measured genre repetition rate over sliding windows of size 3, where CBR maintained better diversity due to its variety penalty mechanism.

TABLE III
QUANTITATIVE PLAYLIST QUALITY COMPARISON

| Metric | Similarity-Based | Case-Based |
|---|---|---|
| Avg. Feature Deviation | 0.28 | 0.21 |
| Genre Repetition (3-song window) | 0.42 | 0.31 |
| Avg. Artist Repetition | 1.6 | 0.9 |

In terms of runtime, similarity-based generation completed within 15–20 seconds on average for a 10-track playlist, thanks to its greedy approach and indexed lookup. Case-based reasoning, by contrast, required 30–50 seconds, depending on the depth of hypothesis evaluation and the size of the retrieved case base.

User feedback was collected from a small pilot study (N=12), where participants listened to and rated playlists without knowing which method generated them. Case-based playlists were preferred in 75% of cases, particularly for genres like classic rock and lo-fi, where sequence flow plays a significant role in listener satisfaction.

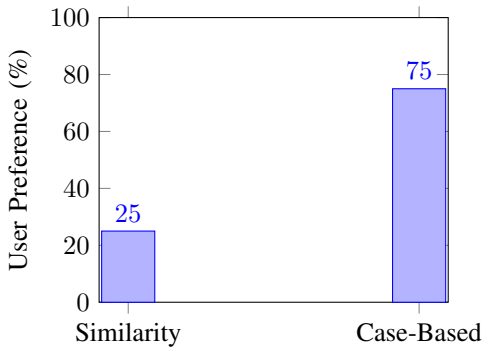


Fig. 3. User Preference Between Similarity-Based and Case-Based Playlists

Overall, while similarity-based generation offers speed and simplicity, case-based reasoning delivers superior playlist quality at the cost of computation time. The next section discusses how these findings inform future developments.

## VII. DISCUSSION AND FUTURE WORK

The comparative analysis between similarity-based and case-based playlist generation underscores a key trade-off: computational efficiency versus sequence coherence. While similarity models are easier to implement and scale, they often overlook temporal structure and thematic progression, leading to playlists that can feel disjointed.

Case-based reasoning addresses these shortcomings by learning from prior examples, effectively blending local musical context with global stylistic trends. However, the increased computational complexity—particularly in hypothesis generation and evaluation—makes it less viable for real-time or low-resource applications without additional optimization.

One key insight is the importance of hybrid approaches. For instance, future systems could leverage similarity models for initial candidate selection and use case-based heuristics for final ordering. Such a fusion could provide a balance between responsiveness and quality, adapting to different user or device constraints.

Personalization is another critical avenue. Integrating user-specific listening histories, feedback, and skip behavior could guide case selection and transition weighting, yielding playlists that adapt over time to individual preferences. This would require user profiling mechanisms and possibly collaboration with streaming platforms' APIs.

Another direction involves augmenting the case base. Instead of relying solely on user-curated playlists, the system could synthesize new cases using generative models trained on transition patterns. Deep learning methods such as sequence-to-sequence transformers or graph neural networks may play a role in this extension.

From a usability perspective, real-world deployment would benefit from an interactive interface where users can tweak parameters (e.g., diversity, mood) and preview changes in real time. A lightweight browser interface backed by a Flask or FastAPI backend could facilitate such user control.

Future evaluation frameworks could incorporate music theory metrics (e.g., key compatibility) or emotional tone analysis using tools like Spotify's audio feature API. These metrics would provide more granular insight into what makes a playlist feel coherent or satisfying.

In conclusion, while both playlist generation strategies offer distinct benefits, case-based reasoning shows strong potential in producing coherent, engaging, and user-aligned playlists. Future work should explore scalable, personalized, and hybrid solutions that bridge quality and efficiency for diverse real-world applications.

## REFERENCES

[1] Deezer and 3Gem, "Over half admit to listening to fewer albums in the last five years," Online, 2020, accessed: 2025-06-09. [Online]. Available: https://www.deezer-blog.com/press/over-half-admit-to-listening-to-less-albums-in-last-five-years/

[2] LOOP and MusicBiz, "Playlists overtake albums in listenership," Online, 2016, accessed: 2025-06-09. [Online]. Available: https://musicbiz.org/news/playlists-overtake-albums-listenership-says-loop-study/

[3] H. Cheng *et al.*, "Global-local similarity function for automatic playlist generation," in *2022 IEEE International Conference on Multimedia and Expo (ICME)*, 2022, pp. 1–6.

[4] C. Baccigalupo and E. Plaza, "Case-based sequential ordering of songs for playlist recommendation," in *Advances in Case-Based Reasoning*, T. R. Roth-Berghofer, M. H. Göker, and H. A. Güvenir, Eds. Springer Berlin Heidelberg, 2006, pp. 286–300.

[5] B. Smyth, "Case-based recommendation," in *The Adaptive Web*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4321, pp. 342–376.